SNAP Data Digitization and Compression with Cric3

Chris Stoughton July 23, 2008; August 6, 2008; June 13, 2009

This note describes software that emulates the digitization and lossy compression of SNAP CCD data. The design of the SNAP mission has readout performed by electronics in the Cric3 board to produce one 16-bit number per pixel. Computing on board will reduce this to 10 (or so) bits per pixel using a lossy compression algorithm. Further computing will apply a lossless compression algorithm to achieve an overall compression factor (from 16 bits per pixel) of 3. The actual compression ratio achieved is the product of the lossy compression ratio and the lossless compression ratio. Choosing the number of bits for lossy compression sets the first ratio: for 16 bits compressed to 10 bits, the compression ratio is 1.6. The compression ratio of the lossless compression depends on the distribution of data values.

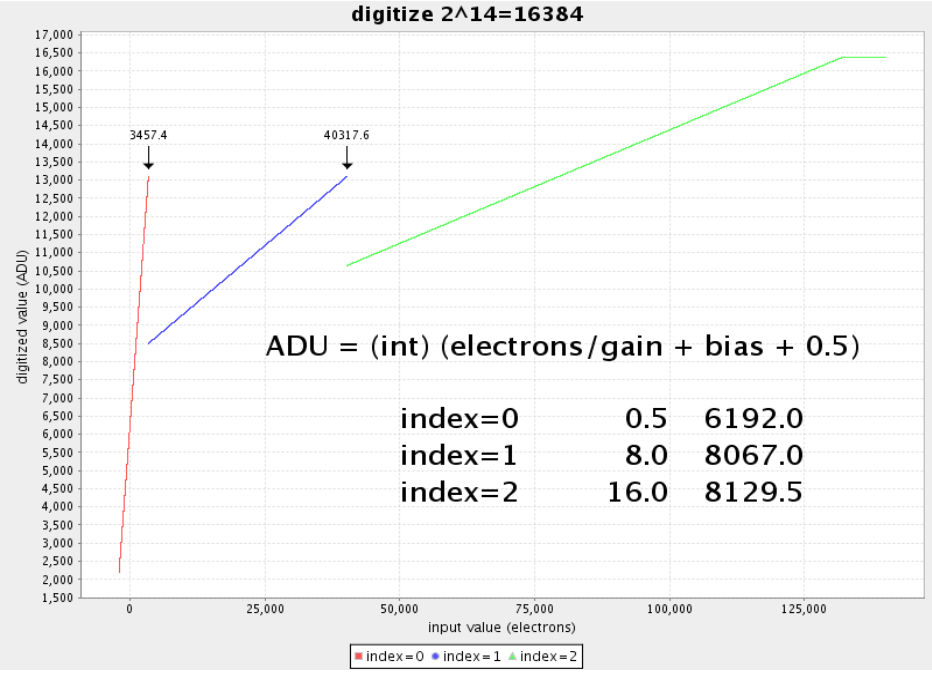
This is by no means the only way to digitize and compress imaging data from space missions. However, it is a convenient way to test the effect of digitization and lossy compression on shape measurement used for weak lensing. The algorithm implemented in hardware in the Cric3 board, and the lossy compression strategy chosen here were designed with these measurement in mind. We could make modest improvements in this implementation by fine-tuning the algorithms, and they can also be de-tuned so they do far worse! However, the default implementation described here serves as an existence proof for mission planners of a conservative estimate of the performance of this strategy at various values of the compression ratio.

The new version of the program adds poisson noise, read noise, and data from a cosmic ray simulation.

Digitization

The number of electrons read out is digitized to Analog-to-Digital Units (ADU) by multiplying by a gain (in electrons/ADU) and adding a bias (in ADUs). The CRIC3 electronics uses one of three pairs of gain and bias, depending on the number of detected electrons in the pixel. The transformation from electrons to ADU is shown in Figure 1.

The gain and bias values are not precisely known yet, but these are reasonable values. The calibration process for SNAP will measure these values in the as-built boards and monitor the values during the mission. The number of electrons at the thresholds (3457.4 between regions 0 and 1; 40317.6 between regions 1 and 2) need to be determined from the as-built boards. These design values are established in boards that have undergone extensive prototyping and testing.



Lossy Compression

The square-root algorithm for lossy data compression described in Gowen and Smith (Rev. Sci. Instr. 2003) transforms an input value to a compressed value as:

$$\text{compressed} = \text{INT}(0.5+a+\text{sqrt}(b*\text{uncompressed}-c)) \qquad (1)$$

where a,b,c are constants specified by the maximum and minimum values of the input and compressed values. For example, compressing Cric3 data to 10 bits yields the values in Table 1 for a,b,c in the three regions

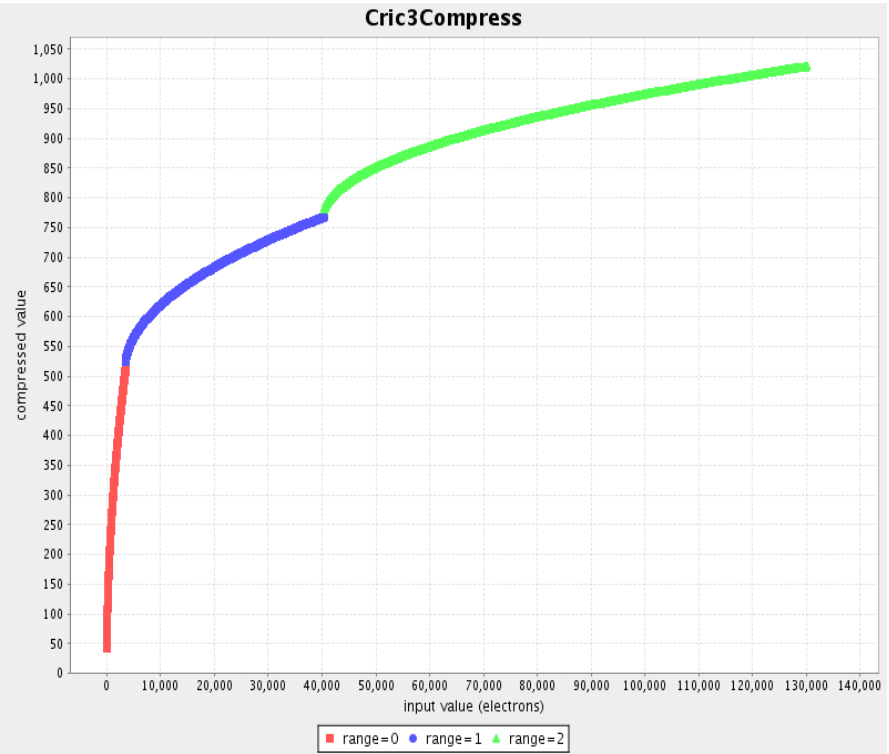
uncompressed		**compressed**		**calculated**		
minimum	maximum	minimum	maximum	a	b	c
6192	13107	0	512	-0.5	37.91	234735.5
8499	13107	512	768	511.5	14.22	120874.7
10649	16384	768	1024	767.5	11.43	121690.1

Table 1. a,b,c values used in 16->10 bit lossy compression of Cric3 data. The minimum values are inclusive and the maximum values are exclusive.

The compression transformation applies eq. (1) with the appropriate values of a,b,c to calculate the compressed value. The decompression algorithm returns the average of all uncompressed values that yield the compressed value determined by eq. (1). The option to smear the decompressed values by randomly selecting one of the uncompressed values that yields the compressed value is in the code, but not enabled by default.

This algorithm can be tuned in two ways. First, the number of bits in the compressed value can be changed. Doubling the compressed minimum and maximum values, for example, will compress to 11 bits. A second tuning option is to reallocate the number of compressed values in each range. The default scheme uses ½ of the available values in low range, and ¼ in each of the other ranges. This allocation can be tuned by changing the compressed minimum and maximum values in each range.

Figure 2 shows the result of the lossy compression as a function of input number of electrons. In range=0, where the number of electrons is low, the mapping is nearly linear. The characteristic "square-root" shape is more evident in ranges 1 and 2.



The result of digitization, lossy compression, and restoration (decompression with averaging and conversion from ADU to electrons) is unbiased, as shown in Figure 3.

Figure 2. The compressed 10-bit value (unitless) vs. the input value (electrons) for compression/decompression.

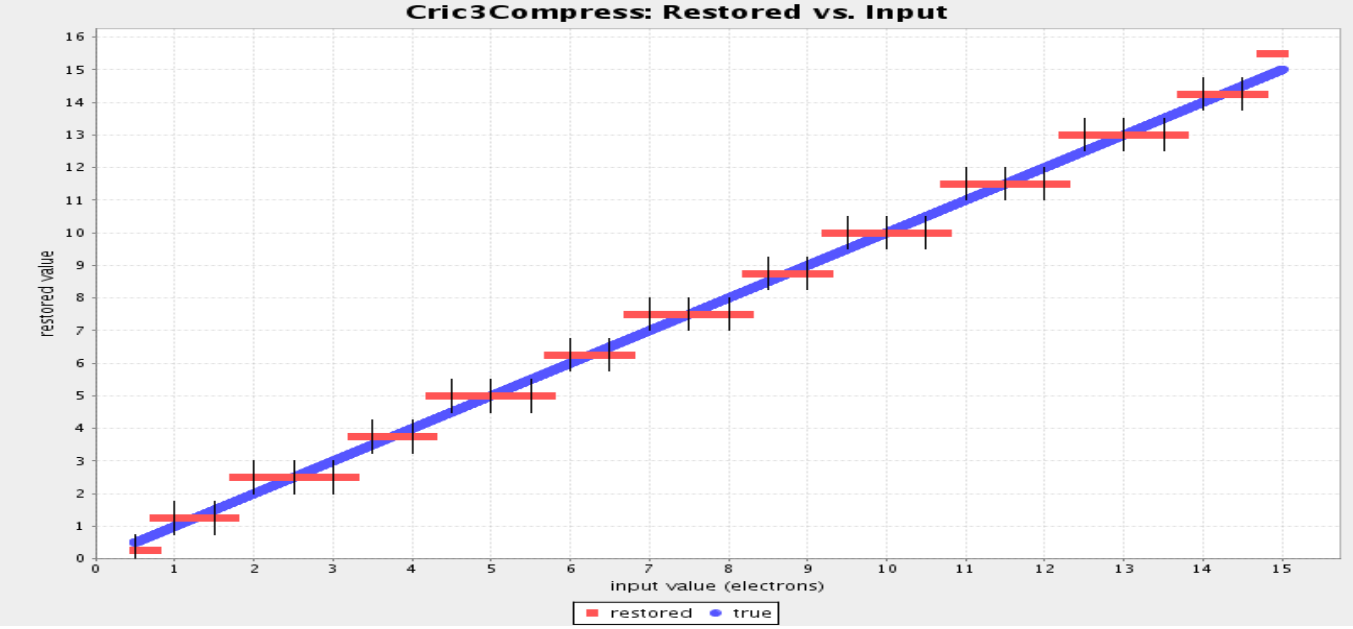


Figure 3. The restored (conversion from electrons to ADU, compressed per Figure 2., uncompressed with averaging, and conversion from ADU to electrons) vs. the input value in electrons.

Note that where multiple input values map to the same compressed value, the restored value is the average of all of the input values. The codec process introduces no bias to a uniform input distribution. The lossy compression *ex vi termini* does not reproduce the input parameters exactly. The difference between the true input value and the codec value has a similar effect on the data as do read noise in the readout electronics or Poisson statistics. Figure 4 illustrates the code noise as a function of input number of electrons.

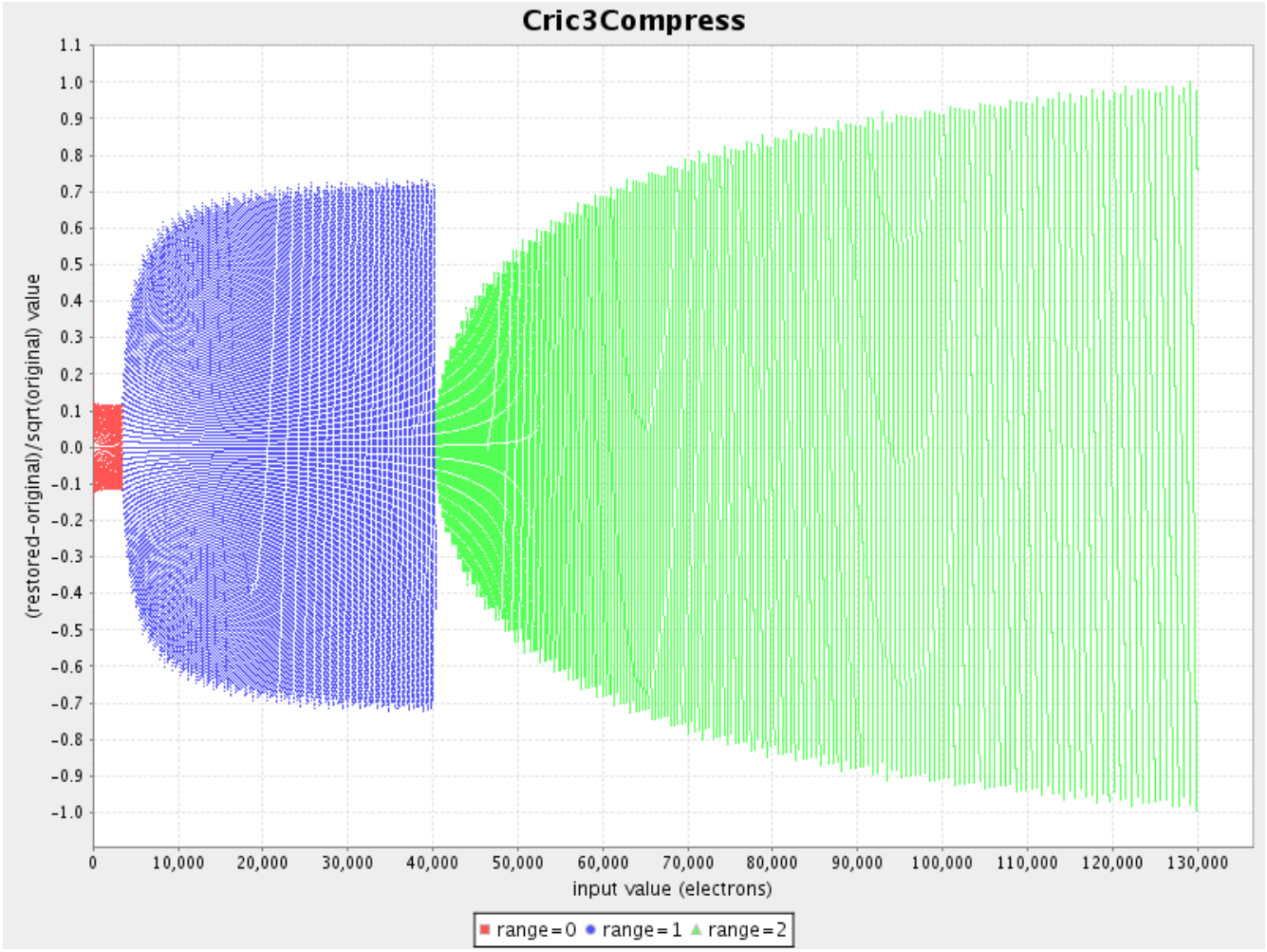


Figure 4. The difference between the restored and original values, as a function of the original input number of electrons. The difference is normalized by the Poisson noise (assuming Gaussian statistics), so values near 1.0 contribute roughly the same amount of noise as Poisson statistics.

For range=0, the extra noise from the codec process is much less than that from Poisson statistics. For brighter objects, the extra noise approaches the noise from Poisson statistics.

Tuning the codec algorithm changes the result shown in Figure 4. Changing the number of bits in the compressed values from 10 to 11 will reduce the difference across the entire range. Increasing the number of compressed values used in a range (by setting the compressed minimum and maximum values in Table 1) reduces the noise in that range. This will increase the noise in another range, where the number of compressed values are decreased by the same amount.

Running the Code

First, you need to get the code. The only prerequisite is to have Sun java 1.6 or higher installed on your computer. You can verify that this is installed with the following command:

$ java -version

java version "1.6.0"

Follow the instructions on the build.snap.nersc.gov site here:
http://build.snap.nersc.gov:2440/reports/cvs/snap.eclipse.help.plugin/html/snap_faq.html#q11

Please note that you need a username/password combination to access the web site and code. The instruction are copied here:

1. Go to http://build.snap.nersc.gov:2440/
2. Click on "All Current Builds" and you will see a list of build named by date and time. In this example, I describe code build on December 12th, 2007 at 11:00:01 PM. You should use the most current version.
3. In the first row, click on snapSim-current-20071204.230001.tgz to download the .tgz file. The exact name depends on the date and time of the build.
4. Untar it (tar xzf snapSim-current-20071204.230001.tgz)
5. This makes a director called snapSim-current-20071204.230001
6. Put that directory in your path, and call that directory LIB_DIR
7. ./eagRunner.sh $LIB_DIR gov.fnal.eag.sim.pixel.Cri3Compress --help

If you run out of memory, add -Xmx1000m after $LIB_DIR to tell it to use more memory.

Note that the following instructions are updated as of June 13, 2008 to reflect a new way to run the code. Here is an example of how I run the code on my laptop.

$ ./eagRunner.sh $LIB_DIR gov.fnal.eag.sim.pixel.Cric3Compress \
/home/stoughto/compressTest/compress_test.fits \

/home/stoughto/compressTest/compress_test_08.fits \

--expTimeKeyword 300 \

--skyPerSecond 0.2 \

--showBernsteinParameter 60,7.7 \

--randomSeedIndex 123 \

--nBits 8 \

--cosmicFileName \

/home/stoughto/snap/chrisb-cosmics/CX_3500x3500x10.5_S4.0_L3.6_N20258_T100_t300.fit \
--doDiagnostics true

Here is what the program does:

1. Read in the input file, which is in units of electrons/second
2. Find the exposure time. If expTimeKeyword is the name of a keyword in the FITS header, use the value of that line as the exposure time. Otherwise, parse the value as a double. In the example, I'm set the exposure time to 300 seconds. Multiply the input values by the exposure time.
3. Multiply the sky rate per second by the exposure time, and add it to the input values.
4. If a file with a cosmic ray exposure is specified, read it in. This file is assumed to be in a total

number of electrons in the exposure, NOT in counts per second, so simply add the values from the cosmic ray exposure to the input file.

5. Seed a poisson random number generator and a normal random number generator with a seed pointed to by the randomSeedIndex. Replace each input value with a random number selected from the random poisson distribution, with mean of the input value. Add a random normal number to each value chosen from a normal distribution, with sigma defined by the read noise.
6. Replace each value with the result of the compress and decompress operation.
7. Write the values to the outputFileName
8. If showBernsetinParameter is true, calculate

   $P_B = \log_2(skySigma / bitsPerStep)$

9. If doDiagnostic is true, plot a histogram of the input pixel values. Also, write the result of this lossy compression followed by lossless huffman encoding. Measure the resulting file size, and calculate the total compression ratio, and write some information to a file named by the outputFileName with a .log suffix.

What does the sky look like?

For an image with mean number of electrons/pixel of 8, digitized with a read noise of 4.0 electrons, the histogram of 1E6 values is shown in Figure 5.

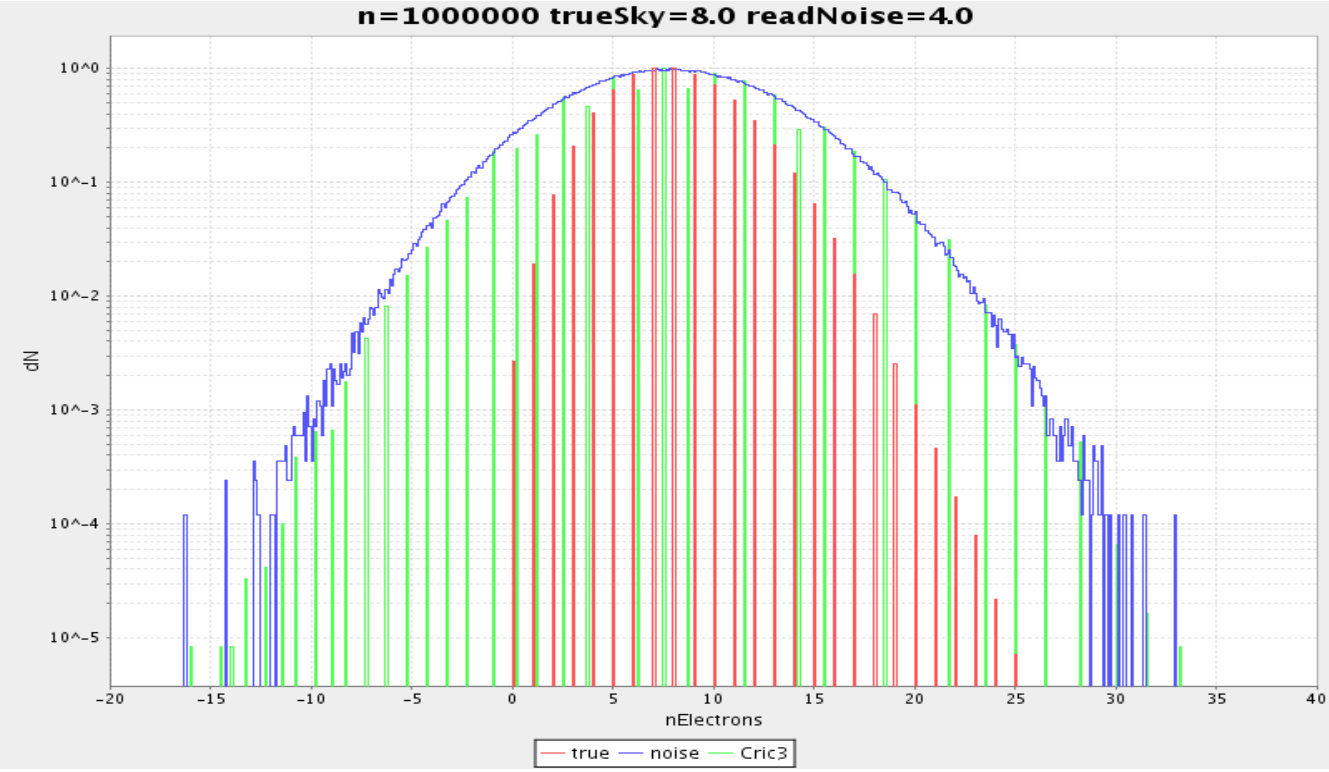


Figure 5. Histogram of sky values: original values; with read noise; and with read noise and codec.

The original values are the red histogram, and are what we would see if each pixel was a perfect photon

counter with no read noise. Sadly, this is not how CCDs work, so what we see before digitization is the blue histogram. After restoring the compressed data, we have the green histogram. Simple sky-finding algorithms rely on the continuity of the blue histogram (no spikes) and fail on codec data. Measuring a simple mean and error in the mean assuming gaussian statistics recovers the true mean sky value without bias, as shown in Table 2.

Histogram	**Calculated Mean**	**Error in Calculated Mean**
original, true values	8.00057	0.00283
with read noise added	7.99847	0.00489
with read noise and Cric3 codec	8.00201	0.00491

Table 2. Calculated sky values for original and codec values

The increase in error of the mean due to Cric3 codec is < 1% of the error increase due to read noise.

What is the compression factor?

The effective compression factor is due to a compression factor from lossy and lossless compression. For lossy compression it does not depend in the input data values. Encoding 16-bit data to 10-bit data has c compression factor of 1.6. A lossless compression algorithm's compression factors depends on the data set. Huffman coding (D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098-1102) constructs a lookup table based on the frequency of input values. Its compression factor is lowest for data with uniformly distributed input values and improves for images where the majority of the input values are similar. This, in general, is the case for astronomical images, where most pixels values are at or near the sky value, with some exceptional pixels having a high value due to galaxies, bright stars, or cosmic ray noise. The Huffman compression factor is measured with images composed of a constant sky level with Poisson statistics, and optionally, one bright star that saturates a few hundred pixels; one UDF image scaled by exposure time; and one cosmic ray image, produced by Chris Bebek, from the set in Table 3.

Cosmic Ray File Name	Landau	# cosmics	thickness (um)	expTime (sec)
blank	--	0	--	--
CX_3500x3500x10.5_S4.0_L0.0_N20258_T200.fit	0.0	20258	200	300
CX_3500x3500x10.5_S4.0_L3.6_N13505_T100_t200.fit	3.6	13505	100	200
CX_3500x3500x10.5_S4.0_L3.6_N13505_T150_t200.fit	3.6	13505	150	200
CX_3500x3500x10.5_S4.0_L3.6_N13505_T200_t200.fit	3.6	13505	200	200
CX_3500x3500x10.5_S4.0_L3.6_N20258_T100_t300.fit	3.6	20258	100	300
CX_3500x3500x10.5_S4.0_L3.6_N20258_T150_t300.fit	3.6	20258	150	300
CX_3500x3500x10.5_S4.0_L3.6_N20258_T200_t300.fit	3.6	20258	200	300

Table 3. Cosmic Ray images used to measure the compression factor. All images have rms diffusion of 4.0 um in 10.5 um pixels. The exposure time sets the number of events.

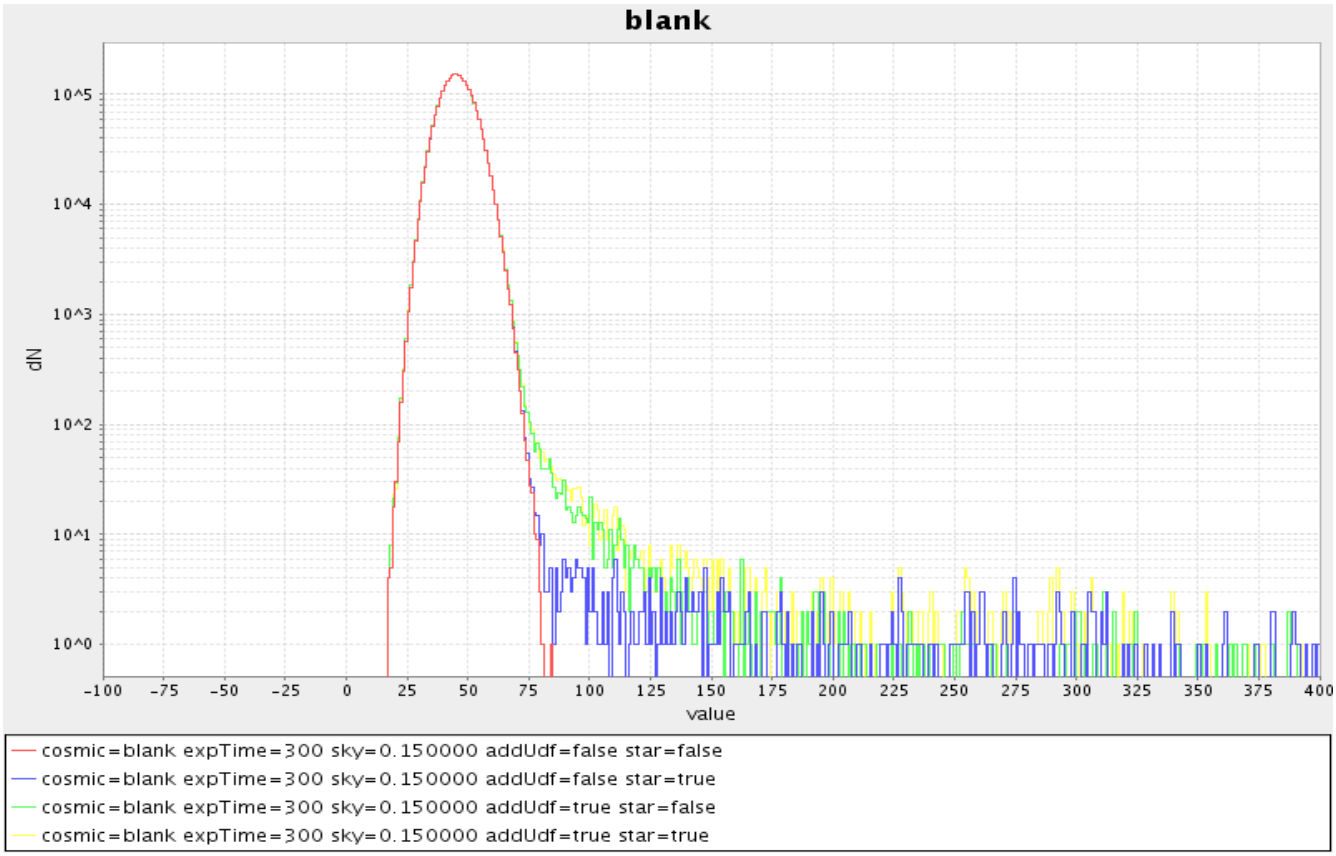


Figure 6: Distribution of pixel values for no cosmic rays, with UDF image and/or one bright star.

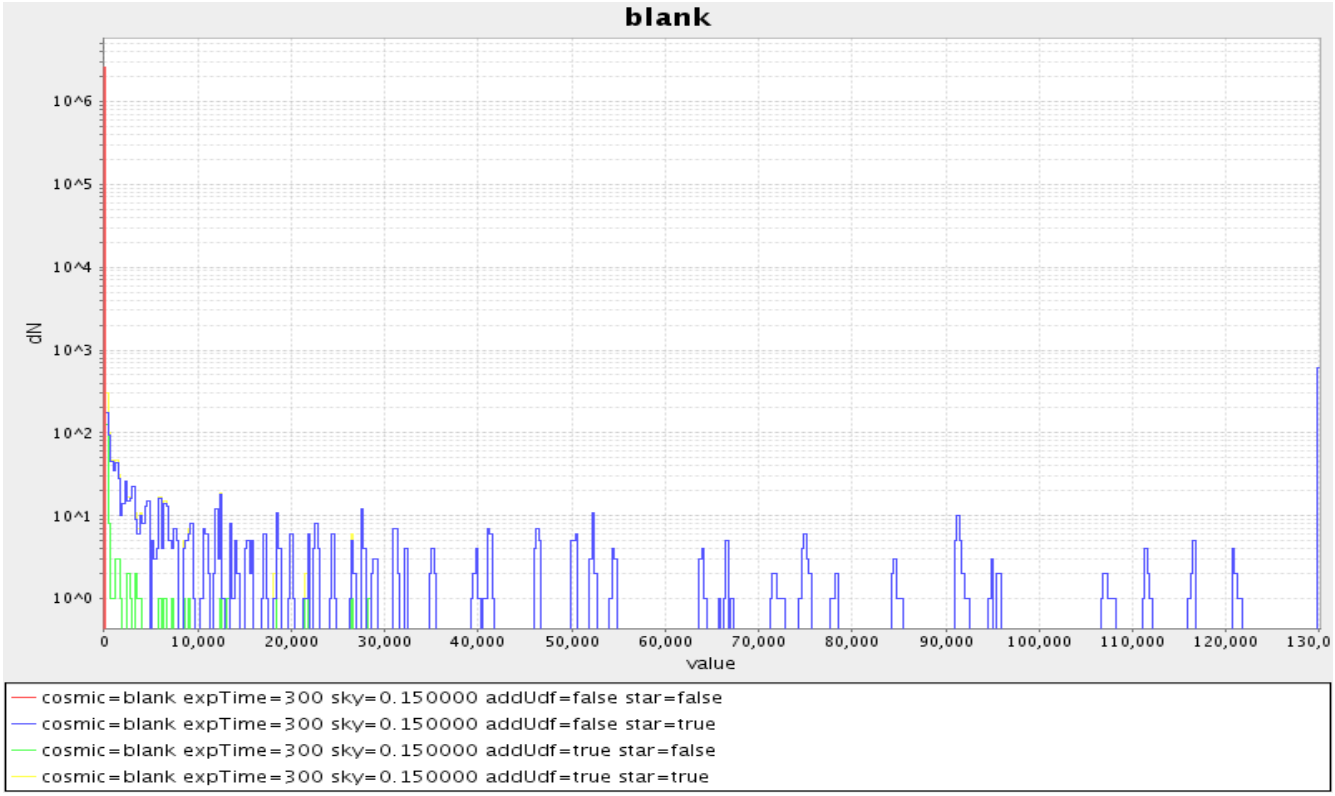


Figure 7: Same as Figure 7, for the entire input range.

Figures 6 and 7 show the distribution of values for an (unrealistic) image with no cosmic rays, with UDF and/or a bright star image included. The compression factors for these four images, for the Cric compression to 10 bits followed by Huffman coding, are in Table 4.

include UDF	include Bright Star	bytes out	compression factor
no	no	1235497	4.1441
no	yes	1239853	4.1295
yes	no	1241734	4.1233
yes	yes	1245762	4.1099

Table 4. Compression factor for an image with no cosmic rays and constant sky level. Adding the UDF image, one bright star, and both images degrades the compression factor.

Cosmic rays dominate the image, increasing the range of values to encode. Figure 8 shows frequency of non-zero values in the cosmic ray files. The resulting compression factors are in Table 5. As expected the image with the most pixels with cosmic ray flux compresses the least.

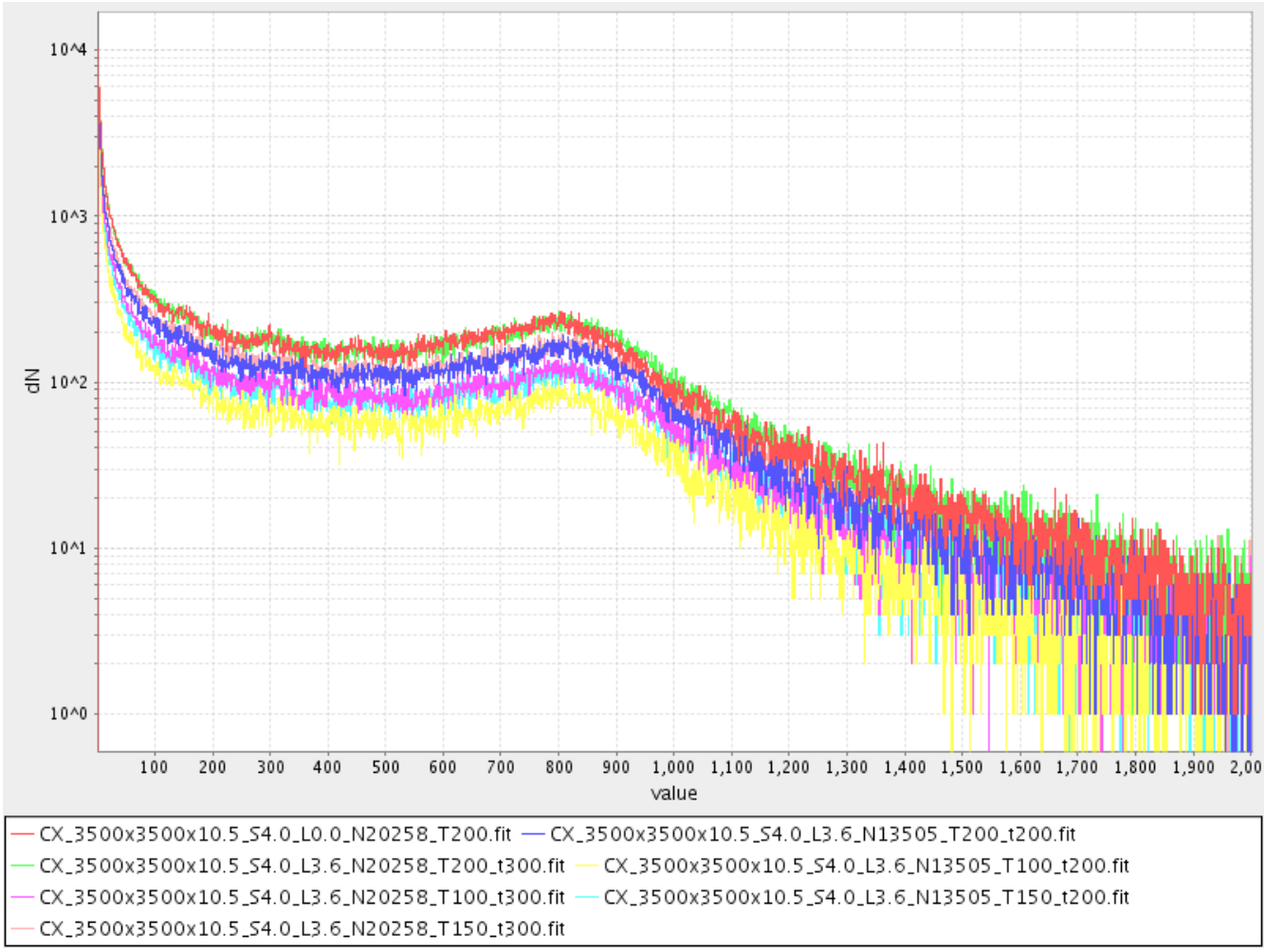


Figure 8. Frequency of non-zero pixel values for the cosmic ray files.

Cosmic Ray File Name	bytes out	compression factor
blank	1245762	4.1099
CX_3500x3500x10.5_S4.0_L0.0_N20258_T200.fit	1520542	3.3672
CX_3500x3500x10.5_S4.0_L3.6_N13505_T100_t200.fit	1370667	3.7354
CX_3500x3500x10.5_S4.0_L3.6_N13505_T150_t200.fit	1404572	3.6452
CX_3500x3500x10.5_S4.0_L3.6_N13505_T200_t200.fit	1456971	3.5141
CX_3500x3500x10.5_S4.0_L3.6_N20258_T100_t300.fit	1414171	3.6205
CX_3500x3500x10.5_S4.0_L3.6_N20258_T150_t300.fit	1469380	3.4845
CX_3500x3500x10.5_S4.0_L3.6_N20258_T200_t300.fit	1525734	3.3558

Table 5: Compression factors for Cric compression to 10 bits followed by Huffman coding, for an image with UDF images and a bright star on the cosmic ray background.

Another variable that impacts the compression factor is the number of bits used in the Cric sqrt algorithm. Varying this from the value of 10 bits used to this point up to 16 bits reduces the compression factor. Table 5 summarizes these values for the least compressible image formed with cosmic ray file name CX_3500x3500x10.5_S4.0_L3.6_N20258_T200_t300.fit.

# Cric compression bits	**bytes out**	**compression factor**
10	1525734	3.3558
11	1840416	2.7820
12	2133343	2.4000
13	2163355	2.3667
14	2180711	2.3479
15	2182839	2.3456
16 (no sqrt transform)	2183560	2.3448